Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № \_

Выполнил студент группы КС-30 Марьин Роман Алексеевич

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/RMaryin\_30/tree/master/Semestr2/Lab%201

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи: 16.03.2021

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать метод сортировки выбором.   
Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), при этом:

* в каждом тесте необходимо по 20 раз генерировать вектор, состоящий из N элементов
* каждый элемент массива заполняется случайным числом с плавающей запятой от -1 до 1, для этого можно использовать как C функцию rand(), так и С++ генераторы:
* каждый массив после генерации необходимо отсортировать и замерить время, требуемое на сортировку, для замера времени использовать следующий код:
* Результат замера для каждой попытки каждого теста записать в файл общий файл.

По окончанию всех тестов необходимо нанести все точки, полученные в результате замеров времени на график, где на ось абсцисс(Х) нанести N, а на ось ординат(Y) нанести значения времени на сортировку. По полученным точкам построить график лучшего (минимальное время для каждого N), худшего (максимальное время для каждого N) и среднего (среднее время для каждого N) случая.

В качестве дополнительного задания, необходимо построить график худшего случая, и график O(c \* g(N)), где g(N) соответствует асимптотической сложности рассматриваемого метода сортировки, подобрав такое значение C, что бы начиная с N ~ 1000 график асимптотической сложности возрастал быстрее чем полученное худшее время, но при этом был различим на графике.

# Описание метода/модели.

Сортировка выбором / Selection sort

На очередной итерации будем находить минимум в массиве после текущего элемента и менять его с ним, если надо. Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах. Асимптотика: O(n2) в лучшем, среднем и худшем случае. Нужно отметить, что эту сортировку можно реализовать двумя способами – сохраняя минимум и его индекс или просто переставляя текущий элемент с рассматриваемым, если они стоят в неправильном порядке. Первый способ оказался немного быстрее, поэтому он и реализован.

Реализация:

void selectionsort(int\* l, int\* r) {

for (int \*i = l; i < r; i++) {

int minz = \*i, \*ind = i;

for (int \*j = i + 1; j < r; j++) {

if (\*j < minz) minz = \*j, ind = j;

}

swap(\*i, \*ind);

}

}

Покажем, почему данная реализация является неустойчивой.

Рассмотрим следующий массив из элементов, каждый из которых имеет два поля. Сортировка идет по первому полю.

Массив до сортировки:

{ (2, a), (2, b), (1, a) }

Уже после первой итерации внешнего цикла будем иметь отсортированную последовательность:

{ (1, a), (2, b), (2, a) }

Теперь заметим, что взаимное расположение элементов (2, a) и (2, b) изменилось. Таким образом, рассматриваемая реализация является неустойчивой.

Так как после каждого прохода по внутреннему циклу делается только один обмен, то общее число обменов равно N-1, что в N/2 раз меньше, чем в сортировке пузырьком.

Число проходов по внутреннему циклу равно N-1 даже в случае сортировки частично или полностью отсортированного массива.

***Наихудший случай:***

Число сравнений в теле цикла равно (N-1)\*N/2.

Число сравнений в заголовках циклов (N-1)\*N/2.

Число сравнений перед операцией обмена N-1.

Суммарное число сравнений N2−1.

Число обменов N-1.

***Наилучший случай:***

Время сортировки 10000 коротких целых чисел на одном и том же программно-аппаратном комплексе сортировкой выбором составило ≈40сек., а ещё более улучшенной сортировкой пузырьком ≈30сек.

Пирамидальная сортировка сильно улучшает базовый алгоритм, используя структуру данных «куча» для ускорения нахождения и удаления минимального элемента.

Существует также двунаправленный вариант сортировки методом выбора, в котором на каждом проходе отыскиваются и устанавливаются на свои места и минимальное, и максимальное значения.

# Выполнение задачи.

1. Подключаем требуемые библиотеки.

2. Генерируем вектор псевдослучайных значений.

3. Пишем функцию сортировки выбором в которую будем передаваться наш вектор.

4. Подключаем кусок кода для замера времени выполнения сортировки.

5. Делаем вывод в файл.

6. Так как требуется выполнить по 20 замеров при каждом количестве исходных значений загоняем большую часть кода в цикл.

#include <iostream>

#include <chrono>

#include <random>

#include <ctime>

#include <vector>

#include <fstream>

#include <cstdlib>

using namespace std;

void sort(vector<double>& data, int lenD) {

int j = 0;

double tmp = 0;

for(int i=0; i<lenD; i++){

j = i;

for(int k = i; k<lenD; k++){

if(data[j]>data[k]){

j = k;

}

}

tmp = data[i];

data[i] = data[j];

data[j] = tmp;

}

}

int main(){

ofstream fout("D:\\Repository\_Maryin\_Kc-30\\RMaryin\_30\\Semestr2\\Lab 1\\speed\_test.txt", ios::app);

fout << endl;

fout << "N = " << 128000;

fout << endl;

fout.close();

for (int g = 1; g < 21; g++) {

double N = 128000;

vector<double> v(N);

setlocale(LC\_CTYPE, "rus");

ofstream fout("D:\\Repository\_Maryin\_Kc-30\\RMaryin\_30\\Semestr2\\Lab 1\\speed\_test.txt", ios::app);

mt19937 engine(time(0));

uniform\_real\_distribution<double> gen(-1.0, 1.0);

for(auto& el: v)

el = gen(engine);

chrono::high\_resolution\_clock::time\_point start = chrono::high\_resolution\_clock::now();

sort(v, N);

chrono::high\_resolution\_clock::time\_point end = chrono::high\_resolution\_clock::now();

chrono::duration<double, nano> nano\_diff = end - start;

chrono::duration<double, micro> micro\_diff = end - start;

chrono::duration<double, milli> milli\_diff = end - start;

chrono::duration<double> sec\_diff = end - start;

if (!fout)

{

cerr << "Ошибка открытия файла!" << endl;

exit(1);

}

fout << endl;

fout << "Попытка №" << g;

fout << endl;

fout << "Time: " << nano\_diff.count() << " nano sec." << endl;

fout << "Time: " << micro\_diff.count() << " micro sec." << endl;

fout << "Time: " << milli\_diff.count() << " milli sec." << endl;

fout << "Time: " << sec\_diff.count() << " sec." << endl;

fout.close();

}

return 0;

}

7. Построил график.

# Заключение.

Сортировка выбором довольно-таки проста, как в своем использовании, так и в алгоритме. Из-за своего алгоритма она является не самой эффективной и если на малом количестве данных она справляется хорошо, то при увеличении количества вводных данных скорость ее работы уменьшается в разы.